Algorithms for the restricted nearest neighbour problem

A Thesis Submitted
in Partial Fulfillment of the Requirements
for the Degree of
Master of Technology

by

Ashish Karkera

to the

Department of Computer Science & Engineering
Indian Institute of Technology, Kanpur

June 2005

Certificate

This is to certify that the work contained in the thesis entitled "*Algorithms for the restricted nearest neighbour problem*", by *Ashish Karkera*, has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

[illegible]

June 2005

(Dr. Sanjeev Saxena)
Professor,
Department of Computer Science & Engineering,
Indian Institute of Technology, Kanpur.

TH

CSE/2005/M
K 145a

Abstract

In this thesis, we propose the following :

1. Simpler algorithms for the Symmetric Angle Restricted Nearest Neighbour (SARNN) problem in the L_1 and L_∞ metrics.
2. Simpler algorithms for the Angle Restricted Nearest Neighbour (ARNN) problem in the L_1 and L_∞ metrics.
3. An optimal $O(n \log n)$ algorithm for the ARNN problem in any L_p metric.

The algorithms in 1 and 2, above, use the plane sweep technique. The previous optimal algorithm for the SARNN problem by Aggarwal and Wee [2] used divide and conquer and the concept of finding the row minimas in a totally monotone matrix, to get the $O(n \log n)$ complexity.

In addition, parallelization of the SARNN problem using the totally monotone matrix approach is discussed.

Acknowledgments

I would like to thank my thesis supervisor, Dr. Sanjeev Saxena, for the guidance that he gave me at key points during this thesis. I would also like to thank my batch-mates, R H Gnanavardhan and Amit Rawat, for the various discussions about my thesis. The help provided by gnana, to increase the clarity of my thesis report, was invaluable.

Contents

List of Tables

List of Figures

Preliminaries

The Angle-Restricted Nearest Neighbour(ARNN) problem and the related problem of finding the Symmetric ARNN(SARNN) graph have applications in wire routing on integrated circuits[6] and in other proximity problems like the relative neighbourhood graph [11]. In this thesis, we propose some algorithms to solve the ARNN and the SARNN problems in various metrics.

1.1 Definitions

First, we give some definitions which will be useful in the rest of the report.

1.1.1 ARNN and SARNN

Given a set $S = \{p_1, p_2, ..., p_n\}$ of n points in the Euclidean plane, we partition the plane around each point into k angular regions where k is a constant.

Definition 1.1 For any particular region, we define the *angle-restricted nearest neighbour (ARNN)* of a point p_i to be a point in the given set that lies in this region and that is the closest to p_i in the L_p metric.

Definition 1.2 Two points, p and q, are *symmetric angle-restricted nearest neighbours(SARNNs)* of each other if p is an angle-restricted neighbour of q in one of its regions and vice-versa.

Definition 1.3 The *ARNN graph* of S has the points of S as vertices and a directed edge from vertex p to vertex q, if and only if q is the ARNN of p in one of its angular regions.

Definition 1.4 The *SARNN graph* of S has the points of S as vertices and an edge between two vertices if and only if the corresponding points are SARNNs of each other.

The ARNN problem consists of finding the ARNN graph of a set of points, S and the SARNN problem consists of finding the SARNN graph of a set of points, S.

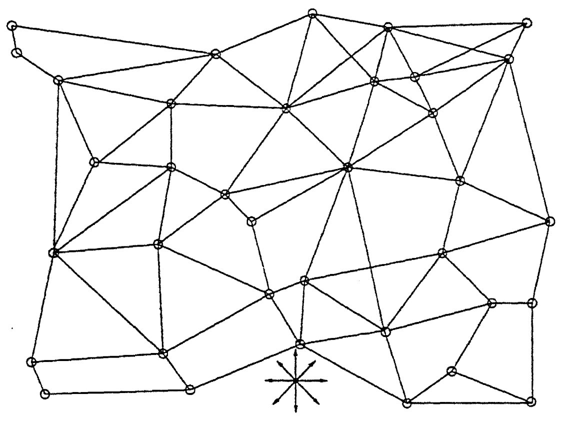

Figure 1.1: An example SARNN graph for $k = 8$

1.1.2 L_p Metrics

The definitions in this subsection are standard and have been taken, almost verbatim, from the web-site[1] [8].

Definition 1.5 The L_p *metric* over $\mathbb{R}^n$ for any $p \geq 1$ is given by equation 1

$$\rho(x, x') = \left[\sum_{i=1}^{n} |x_i - x'_i|^p \right]^{1/p} \tag{1}$$

For each value of p, equation 1 is called an L_p metric. The three most common metrics are L_1, L_2 and L_∞.

[1] *http://msl.cs.uiuc.edu/planning/node177.html*

Definition 1.6 L_1 : The *Manhattan metric* , which is often nicknamed this way because in $\mathbb{R}^2$ it corresponds to the length of a path that is obtained by moving along an axis-aligned grid. Taking $p = 1$ in Equation 1, for two points $p(x, y)$ and $q(x', y')$, we get $DIST_{L_1}(p, q) = |x - x'| + |y - y'|$.
Example The L_1-distance from (0,0) to (2,5) is 7 by traveling "east two blocks" and then "north five blocks" ∎

Definition 1.7 L_2 : The *Euclidean metric* , which is the familiar Euclidean distance in $\mathbb{R}^n$.

Definition 1.8 L_∞ : The L_∞ metric must actually be defined by taking the limit of Equation 1 as p tends to infinity. The result is :

$$L_\infty (x, x') = max_{1 \leq i \leq n} |x_i - x'_i| \tag{2}$$

1.1.3 Monotonicity

Definition 1.9 An array is called *monotone* if the minimum entry in the i^{th} row lies below or to the right of the minimum entry in the $(i-1)^{th}$ row for all $2 \geq i \geq n$.

Definition 1.10 An array is said to be *totally monotone* if every one of its 2×2 sub-arrays is monotone.

Definition 1.11 A *sub-array* of an array A is obtained by deleting one or more rows and/or columns of A.
Example If we have a 5×5 array and we delete any one row and any two columns from it, we are left with a 4×3 sub-array of the original array. ∎

1.2 Organisation of the Thesis

In Chapter 2, we first describe the sequential algorithm for the SARNN problem and then provide bounds for the parallelization of the same problem using the totally monotone matrix approach. In Chapter 3, we describe two simple algorithms for the SARNN(Symmetric Angle Restricted Nearest Neighbour) problem in the L_1

and L_∞ metrics. In Chapter 4, we see how minor modifications to the algorithms of Chapter 3 can be made to solve the ARNN(Angle Restricted Nearest Neighbour) problem in the L_1 and L_∞ metrics. In Chapter 5, we propose some modifications to Aggarwal and Wee's algorithm [2] for the SARNN problem, to optimally solve the ARNN problem for any L_p metric, $1 < p < \infty$, in $O(n \log n)$ time.

Chapter 2

The SARNN problem and its parallelization

2.1 Sequential algorithm to find the SARNN graph

The sequential bound for the SARNN problem is $O(n \log n)$ which was proved by Aggarwal and Wee in [2]. Their algorithm makes use of two techniques :

1. Divide and conquer and

2. Finding the row minimas in totally monotone matrices.

Let us assume that $k = 8$. Hence, the plane around each point is to be divided into eight regions (see Figure 2.1). We pair opposite regions and solve the SARNN problem for each of these pairs separately.

Suppose, for example, we consider the pair of regions EN and WS (see Figure 2.1). Then, the sequential algorithm [2] for finding the ENWS SARNN graph of a set of points, P, is as follows:

1. Divide the given n points into roughly equal parts, U and V, by a line of slope 1, U and V, where U comprises of points below the dividing line and V comprises of points above the dividing line. Also, $|U| = r$ and $|V| = q$. This is done by first computing $c_i = y_i - x_i$ for every point (x_i, y_i), after which the median of the c_i s is found (say c). Then, the dividing line will be $c = y - x$.

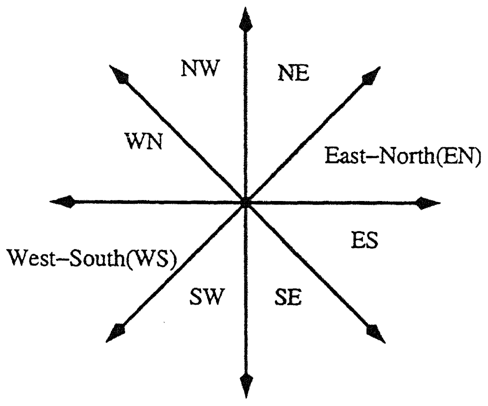

Figure 2.1: The regions around a point (for $k = 8$)

2. Compute the ENWS(U) and ENWS(V) recursively using the same procedure.

3. A pair of points $u_i \in U$ and $v_j \in V$ are said to form a *restricted symmetric pair* for U and V if u_i and v_j are symmetric angle-restricted nearest neighbours in P. We compute the restricted symmetric pairs of U and V as follows :

   (a) Convert this problem into that of finding the minimum entry of each row of a totally monotone matrix.

   > This is done by making the i^{th} row represent the i^{th} point of U and the j^{th} column represent the j^{th} point of V. Every entry of the matrix represents the L_p distance between the point represented by its row and the point represented by its column.

   The matrix formed will be totally monotone. Here, we note that the entire matrix does not need to be computed in advance. Every entry of the matrix can be computed, on the fly, by a single processor in constant time.

   (b) We find the row minimas of this matrix using the algorithm of [1]. Similarly, we find the column minimas.

   (c) If an entry of the matrix is the minimum in its row and in its column then the point representing the row and the point representing the column form a restricted symmetric pair.

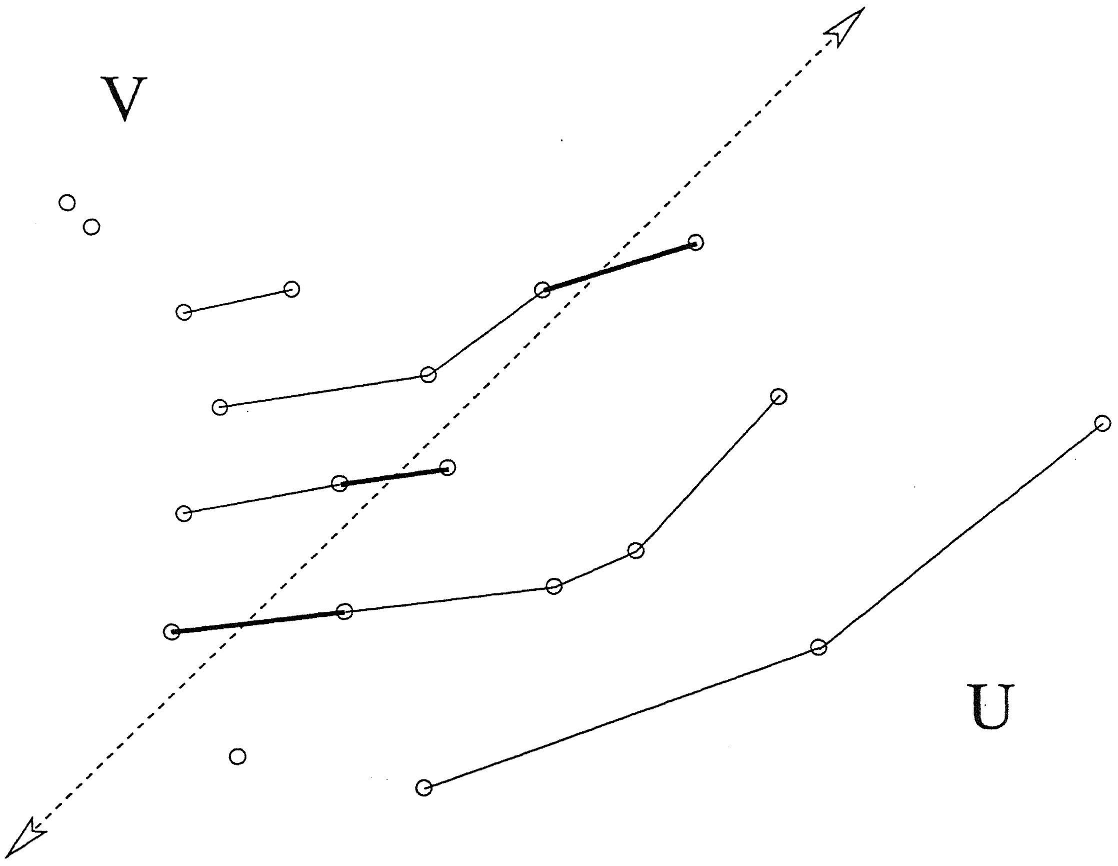

Figure 2.2: Restricted Symmetric pairs (joined by dark lines)

2.1.1 Analysis

The recurrence relation is : $T(n) = 2\,T(n/2) + D(r, q)$
where $D(r, q)$ is the time to compute the restricted symmetric pairs of U and V. As the algorithm of [1] takes $O(n)$ time for computing $D(r, q)$, we have $T(n) = O(n \log n)$. Also, space taken is $O(n)$ as the matrix does not need to be computed in advance. Instead, the entries of the matrix are calculated as and when needed in $O(1)$ time each.

2.2 Parallel bounds for the SARNN problem

Here, we parallelize the algorithm that was described in the previous section.

2.2.1 Parallel Algorithm

In step 1 of the algorithm, the points are separated into two halves. This is done by finding the median of the $(y-x)$s of all the points. This step takes $O(\log n \log\log n)$ time using a linear number of operations [7]. Another way would be to sort the c_is $= (y_i - x_i)$s in $O(\log n)$ time and $O(n \log n)$ operations [7]. This sorting will also be useful in the rest of the stages. Next, we solve the problem for the individual groups recursively in parallel. The final step is finding the restricted symmetric pairs. This involves finding the row minimas of the totally monotone matrix formed. Let us assume that this takes $f(n)$ time and $n/g(n)$ processors. Obviously if $f(n) = g(n)$ then the algorithm is work optimal and if $f(n) = g(n) = \log(n)$ then the algorithm is both time and work optimal.

2.2.2 Analysis

In stage 1 of the algorithm we have an $(n/2) \times (n/2)$ monotone matrix. Finding the row minimas of this matrix takes $f(n)$ time and uses $n/g(n)$ processors. Hence, the work done in this step is $(n \times f(n))/g(n)$. Similarly, in stage 2 of the algorithm we have 2 $(n/4) \times (n/4)$ matrices and we have to find their row minimas using $n/(g(n) \times 2)$ processors each. Hence, totally $(n \times f(n/2))/g(n/2)$ work is done in this step. This continues till the matrices have a fixed constant number of elements in them. In each stage, both sides of the matrix are halved. Hence, there are $O(\log n)$ steps in this algorithm.

$$\text{Work} = n\left\{\frac{f(n)}{g(n)} + \frac{f(n/2)}{g(n/2)} + \frac{f(n/4)}{g(n/4)} + \cdots (\log n \text{ terms})\right\}$$

We also know the following :: $f(n) \geq g(n)$, $f(n) \geq f(n/2)$ and $g(n) \geq g(n/2)$.

$$\therefore \frac{f(n)}{g(n)} \geq \frac{f(n/2)}{g(n/2)} \geq \frac{f(n/4)}{g(n/4)} \geq \cdots$$

$$\therefore \text{Work} = O\left(n\left\{\frac{f(n)}{g(n)} + \frac{f(n)}{g(n)} + \cdots (\log n \text{ terms})\right\}\right)$$

$$\text{Work} = O\left(\frac{n \times f(n) \times \log n}{g(n)}\right)$$

Similarly, we get,

$$\text{Time} = O(f(n) \times \log n)$$

Thus, the time taken to compute the SARNN graph in parallel(using the monotone matrix paradigm and the CREW model) is at most $O(\log n)\times$ the time taken by the black-box routine. The work complexity is also increased by a factor of $O(\log n)$.

The best known time-optimal parallel algorithm for the row minima in totally monotone matrix is from [3] which does it in $O(\log n)$ time. Using this, the parallel algorithm for the SARNN problem will take $O(\log^2 n)$ time and $O(n \log^2 n)$ work. Similarly, the best known work-optimal parallel algorithm for the row minima in totally monotone matrix is from [4] which does it in $O(\log n\sqrt{\log n \log\log n})$ time. Using this, the parallel algorithm for the SARNN problem will take $O(\log^2 n\sqrt{\log n \log\log n})$ time and $O(n \log n\sqrt{\log n \log\log n})$ work. A theoretical time and work optimal parallel algorithm for the row minima problem would take $O(\log n)$ time and $O(n)$ work. Hence the parallel algorithm for the SARNN problem would take $O(\log^2 n)$ time and $O(n \log n)$ work, i.e. would be work-optimal. /fussy

As yet, there is no NC (poly-logarithmic time), work optimal($O(n)$) algorithm for finding the row minimas in a totally monotone matrix.

Type	Model	Time	Work	Reference
Sequential	-	$O(n)$	-	[1]
Parallel	CRCW	$O(\log n \log\log n)$	$O(n\sqrt{\log n})$	[4]
	CREW	$O(\log n(\log\log n)^2)$	$O(n\sqrt{\log n})$	[4]
	EREW	$O(\log n)$	$O(n \log n)$	[3]
		$O(\log n\sqrt{\log n \log\log n})$	$O(n\sqrt{\log n \log\log n})$	[4]
Randomized Parallel	EREW	$\tilde{O}(\log n)$[1]	$O(n)$	[10]
	CRCW	$\tilde{O}(\log\log n)$	$O(n)$	[10]

Table 2.1: Complexities of various algorithms for finding the row minimas in a totally monotone matrix

[1] $\tilde{O}(f(n))$ denotes that, with high probability, the complexity is $O(f(n))$.

Chapter 3

A simple algorithm for the SARNN problem

In this chapter, we present simpler algorithms for the SARNN problem in the L_1 and L_∞ metrics. We use the plane sweep technique of [5]. In the plane sweep data structure, we store the points which could be the SARNN for some future event point. Any point, which cannot be a SARNN for a future event point, is deleted from the data structure.

Here too, like in [2], we assume that $k = 8$. i.e. the plane around each point is to be divided into eight regions. We pair opposite regions and solve the SARNN problem for each of these pairs separately. Suppose, for example, we consider the pair of regions EN and WS. First, we solve the problem for the L_1 metric.

3.1 SARNN in the L_1 metric

We first sort the points in the direction of the line with slope 1. i.e sort them by the sum $(y_i + x_i)$. Then, we proceed from the point which has greatest value of $(y_i + x_i)$, towards the point which has lowest value of $(y_i + x_i)$. Each point will now be called an *event point.* The sweep line has slope -1. We assume that the sweep line intersects only 1 point at a time i.e. $\forall i$, $(x_i + y_i)$ is unique. The sweep line stops at each event point, does some processing and then moves on to the next event

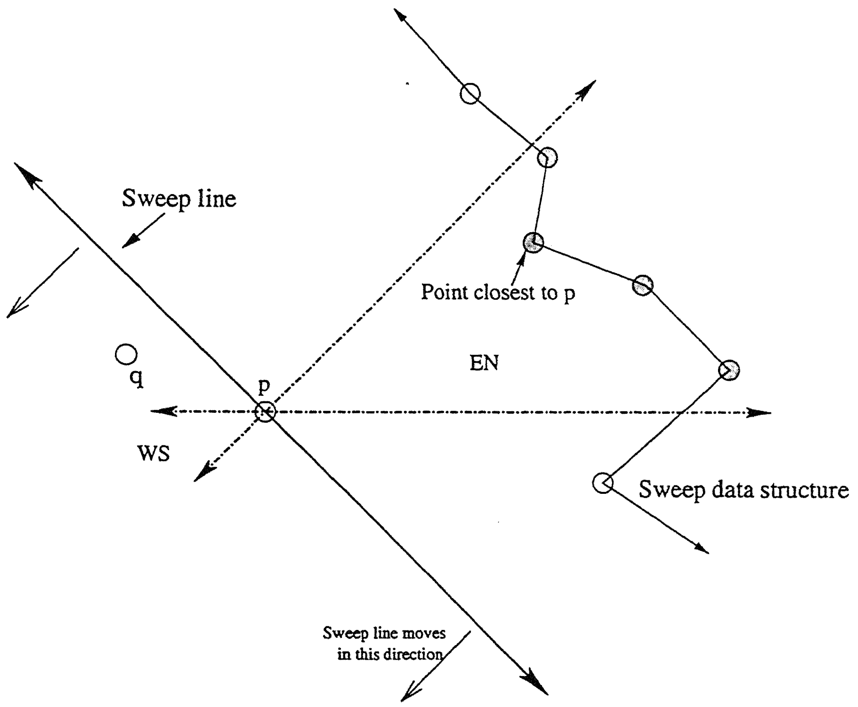

Figure 3.1: Sweep line and data structure at event point P

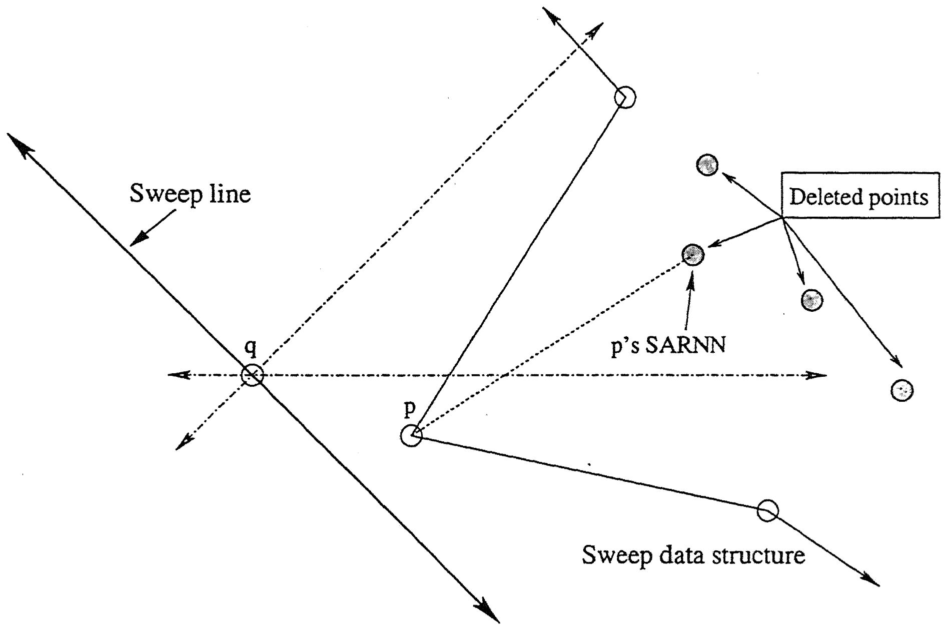

Figure 3.2: Sweep line and data structure after processing point P

ınt. At each event point, we find the set of points (the shaded points in Fig. 3.1) the data structure which could be the SARNN for the event point. i.e. which lie the EN region of the event point. Assuming that the event point is (x_i, y_i) and point to be checked is (x_j, y_j), this can be checked by the following 2 conditions:

1. $y_j > y_i$

2. $(y_i - x_i) > (y_j - x_j)$

Each such point(e.g. the shaded points in Fig. 3.2) is deleted from the structure d then the L_1 distance of this point from the event point is found. If, it is less an the current known smallest distance of the points seen so far, then this point assumed, for the time-being, to be the SARNN of the event point in the EN gion. After all the points in the EN region of the event point have been looked at d deleted, the event point is added to the data structure and then the sweep line oceeds to the next event point.

This is done until the sweep line goes past the last point. At this stage, the ints still on the data structure don't have a neighbour in the WS direction and nce, they can be deleted. The correctness of this method follows from Lemma 3.1.

emma 3.1 *For any point q, which lies in the data structure and is in the EN gion of the event point p, the ARNN of q, in the WS direction, in the L_1 metric, p.*

roof: Refer to Fig. 3.3. Draw a horizontal line-segment from q to the sweep ne, which meets the sweep line at point s. Draw another line-segment from q, rpendicular to the sweep line and which meets the sweep line at r. Any point ithin the isosceles right-angled Δqrs will have an L_1 distance from q which is less an the L_1 distance between p and q. Suppose that such a point, r, exists.

We can see that r lies in the WS region of q. Hence, it will be the ARNN of q the WS region. But, r would have been a previous event point and since q lies in s EN region, q would have been deleted from the data structure. We know that q

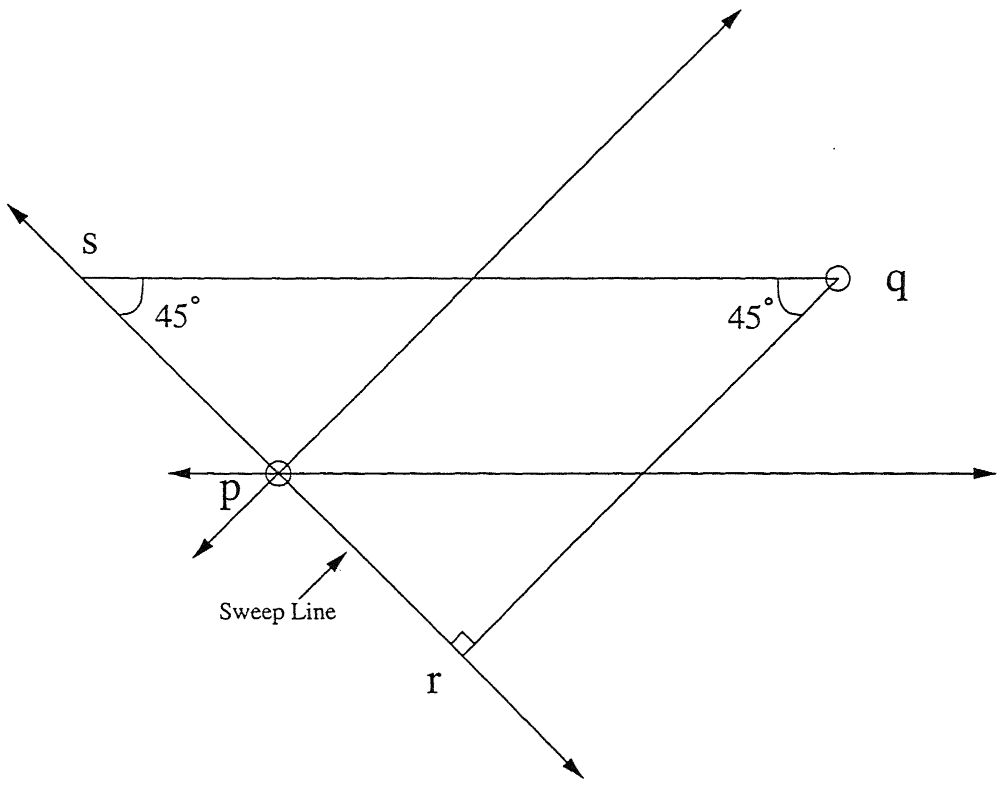

Figure 3.3: An illustration of the proof of Lemma 3.1

is still in the sweep data structure. Hence, the hypothetical point, r, can not exist. Hence, p is the ARNN for any such point q in the L_1 metric. ∎

Because of the above lemma we know the following:
(a) Only such points, whose ARNNs in the WS direction in the L_1 metric have been found, are deleted.
(b) The SARNN of the event point in the EN region is found correctly.

3.2 SARNN in the L_∞ metric

For the L_∞ metric, we have a lemma which is analogous to Lemma 3.1. Here too, we assume that the sweep line intersects only one point at a time.

Lemma 3.2 *For any point q, which lies in the data structure and is in the EN* *on of the event point, p, the ARNN of q, in the WS direction of q, in the L_∞ ric, is p.*

of: Refer to Fig. 3.4. Draw a horizontal line-segment from q to the sweep meeting it at s. Draw another line-segment from q to the sweep line making ıngle of 45° at q and which meets the sweep line at r, a point lower than s. Any ıt within the isosceles right-angle Δqrs will be in the WS region of q and will ɜ an L_∞ distance from q which is less than the L_∞ between p and q. Suppose ; such a point, r, exists.

Hence, r will be the ARNN of q in the WS region. But, r would have been a ᵣious event point as the sweep line proceeds from right to left, and since q lies in ƐN region, q would have been deleted from the data structure. We know that q ;ill in the sweep data structure. Hence, the hypothetical point, r, can not exist. ıce, p is the ARNN of any such point q in the L_∞ metric. ∎

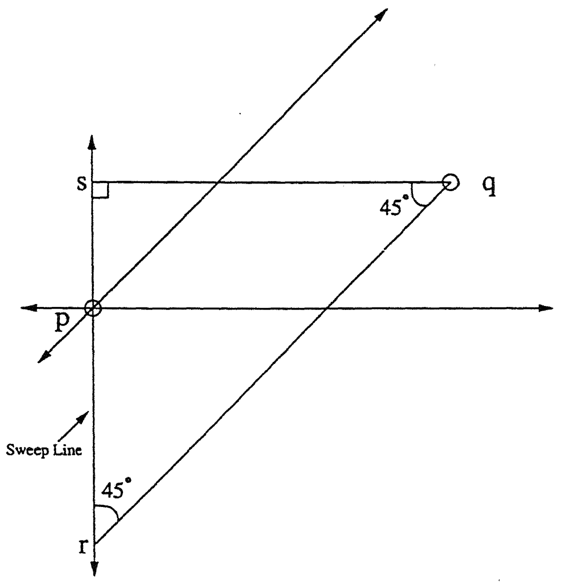

Figure 3.4: An illustration of the proof of Lemma 3.2

We use the same technique as for the L_1 metric except for a few minor changes. ɜre, the points are sorted according to their X-coordinates. Also, the sweep line vertical and it moves horizontally from right to left. For every event point, the ıints in the sweep data structure, which are also in its EN region are looked at and evaluated according to their L_∞ distance from the event point. We see that all the points in the data structure in the EN region of the event point can be deleted from it (after checking the L_∞ distance from the event point) because of Lemma 3.2 above. The full algorithm, for the sake of completeness, is given in Appendix A.

3.3 Complexity Analysis

The points can be sorted in $O(n \log n)$ time in the required direction. At each event point we find the lowest point on the data structure which could be the event point's SARNN. This can be found in $O(\log n)$ time as we can use a red-black tree (see Cormen et al. [5]) to store the sweep data structure. This is done for each of the points in the set. Hence this part of the algorithm takes $O(n \log n)$ time.

Next, we notice that points are added only once to the sweep data structure and deleted when their distances to an event point are found. Hence there are $O(n)$ updates to the data structure in total. As each insertion or deletion in a red-black tree takes $O(\log n)$ time [5], it follows that the overall algorithm takes $O(n \log n)$ time.

Chapter 4

A simple algorithm for the ARNN problem

In this chapter, we present simpler algorithms for the ARNN problem in the L_1 and L_∞ metrics. The techniques and data structures used are the same as that for the SARNN problem.

Here too, like in [2], we assume that $k = 8$. Hence, the plane around each point is to be divided into eight regions. However, we don't pair regions like in the SARNN problem. Instead, the graph for each region is found separately and then merged later on to get the complete ARNN graph.

4.1 ARNN in the L_1 metric

The algorithm for the ARNN problem in the L_1 metric is not very different from the algorithm for the SARNN problem in the L_1 metric. The only change in the SARNN algorithm is that at each event point, we set the point in the sweep data structure which was the closest to the event point in its EN region to be the SARNN of the event point. Here, we set the ARNN in the WS direction of all points in the EN region of the event point, and in the data structure, to be the event point. All such points are deleted from the data structure after this, the event point is then added to the data structure and the sweep line proceeds to the next event point.

the event point is then added to the data structure and the sweep line proceeds to the next event point. The full algorithm for the sake of completeness is given in Appendix B.

4.2 ARNN in the L_∞ metric

The algorithm to find the ARNN graph in the L_∞ metric is the same that for the SARNN graph in the L_∞ metric except for the changes mentioned above. The full algorithm for the sake of completeness is given in Appendix C

4.3 Analysis

There is not much change in the algorithms from that of the SARNN graph. We look at each of the points in the data structure once, immediately set its ARNN in the WS region and then delete it. Hence, the data structure is updated $O(n)$ times. As each update to the data structure takes $O(\log n)$ time [5], the complexity remains $O(n \log n)$.

Chapter 5

An optimal algorithm for the ARNN problem

Guibas and Stolfi [6] describe a divide and conquer algorithm to solve the *Angle-Restricted Nearest Neighbour(ARNN)* problem that takes $O(n \log n)$ time for the L_1 and L_∞ metrics. For the general case of the L_p metric where $p \geq 1$, Wee et al. [11] describe a divide and conquer algorithm that takes $O(n \log^2 n)$ time. Here, we present an algorithm to solve the ARNN problem in the L_p metric for $p \geq 1$ in $O(n \log n)$ time.

5.1 The Algorithm

The algorithm that we use for the ARNN problem is almost the same as that used by Aggarwal and Wee [2] for the SARNN problem except for some changes which are described below. First, we compute the L_1 nearest neighbour for each point in all the regions. This we can do by using the method in [6] or by using the algorithm proposed in section 4.1 earlier.

We compute the ARNN graph for each region separately. Here, we compute the ARNN graph of the set of points, S for only the West-South(WS) region, which we shall denote by WS(S). It is easy to see that if an algorithm finds the ARNNs of all points in the WS region then by suitable rotation of the axes, we can find the

ARNNs of all the points in all of the regions and hence obtain the complete ARNN graph.

We have a set S of points for which we need to find the ARNN graph. Let $\bar{l}$ denote a line with slope 1, that splits the points in S in two roughly equal parts, $U = \{u_1, \cdots, u_r\}$ and $V = \{v_1, \cdots, v_q\}$, the points of which occur in increasing y-coordinates, the points of U lie to the right of $\bar{l}$ and the points of V lie to its left. Now to compute WS(S). We individually compute WS(U), WS(V) and the *restricted pairs* for U and V. Hence, we need to compute all the restricted pairs of U and V.

Definition 5.1 A pair of points, $u_i \in U$ and $v_j \in V$ form a *restricted pair* if v_j is the angle-restricted nearest neighbour of u_i in the WS region.

To compute the restricted pairs, we transform this problem into that of finding the row minimas in a totally monotone matrix, in linear time. Now, since finding the row minimas in a totally monotone matrix also takes linear time using the algorithm in [1], we can solve the overall problem in $O(n \log n)$ time which is optimal.

Definition 5.2 A point v is *dominated* by a point u if v lies in the West-South octant of u. In other words, if the coordinates of u and v are (x_u, y_u) and (x_v, y_v) respectively, then $y_u > y_v$ and $(y_u - x_u) > (y_v - x_v)$.

Lemma 5.1 *Let $u_i, u_j \in U$ be any two points such that u_j dominates u_i, and let v_k be any point in V whose y-coordinate is no higher that that of u_i. Then, v_k cannot be the restricted nearest neighbour of u_j among all points in S.*

Proof: See proof of Lemma 2.1 in [2]. ▮

Let the *L_p nearest neighbour* of a point r be the point closest to r if the L_p metric is used to compute the distances.

Lemma 5.2 *If the L_1 nearest West-South neighbour of a point $u \in U$ belongs to U, then the L_p nearest West-South neighbour of u also belongs to U, for any $p > 1$.*

Proof: See proof of Lemma 2.2 in [2]. ∎

Let $U' = \{u_i \in U|$ the L_1 nearest West-South neighbour of u_i belongs to $V\}$ and let RP(U,V) denote the restricted pairs for U and V. Then from Lemma 5.2, it follows that RP(U, V) = RP(U', V). Hence, from now on, we compute RP(U', V) instead of RP(U, V).

Lemma 5.3 *Let $u_i, u_j, u_l \in U'$ such that u_j dominates u_i and the y-coordinate of u_l is no less than that of u_j. And, let v_k be any point in V whose y-coordinate is less than that of u_i. Then, u_l cannot be the L_p nearest neighbour of v_k in the EN region.*

Proof: See proof of Lemma 2.3 in [2]. ∎

5.1.1 Transformation

Next, we transform the problem of finding the restricted pairs into that of finding the row minimas in a totally monotone matrix. We first note that the points of $U' = \{u_1, \cdots, u_r\}$ and of $V = \{v_1, \cdots, v_q\}$ occur in the increasing order of their y-coordinates. The next two definitions are taken from [2].

Definition 5.3 For $1 \leq i \leq q$, the *top anchor* of u_i, denoted by *top(u_i)*, is defined to be the index of the highest point of V that belongs to the West-South octant of u_i. Formally, top(a) = b if $y_b = \max\{y_j \,|\, y_j < y_a \text{ and } j \in V\}$.

Definition 5.4 For $1 \leq i \leq q$ the *bottom anchor* of u_i, denoted by *bot(u_i)*, is defined as follows :

$$bot(u_i) = \begin{cases} bot(u_{i-1}) & \text{if } u_i \text{ does not dominate } u_{i-1} \\ top(u_{i-1}) & \text{otherwise} \end{cases}$$

We can find the top and bottom anchors for all points in U' in $O(r+q)$ time [2].

Lemma 5.4 *For $1 \leq i \leq r$ and $1 \leq j \leq q$, u_i can be the restricted nearest neighbour of v_j only if*

(a) $j \leq top(u_i)$ and

(b) $j > bot(u_i)$.

Proof: If $j > top(u_i)$ then v_k does not lie in the West-South octant of u_i. Inequality (b) follows from Lemmas 5.1 and 5.3. ∎

Now, we use the above lemma in constructing the totally monotone matrix. We consider a $r \times q$ matrix, $M = m(i,j)$, such that for $1 \leq i \leq r$ and $1 \leq j \leq q$, $m(i,j)$ equals the L_p distance, $p > 1$, between u_i and v_j if both inequalities (a) and (b) of Lemma 5.4 hold, and $m(i,j) = \infty$ otherwise. Then, for $1 \leq i \leq r$ and $1 \leq j \leq q$, (u_i, v_j) can form a restricted pair for U' and V only if $m(i,j)$ is the smallest entry in the i^{th} row.

Lemma 5.5 *For* $1 \leq i \leq r$, *let* $s(i)$ *be defined such that* $M(i, s(i))$ *contains the minimum entry in the* i^{th} *row of M. Then,* $\forall 1 \leq i \leq r$, $s(i)$ *can be computed in* $O(r+q)$ *time.*

Proof: See proof of Lemma 3.2 of [2]. In the proof of Lemma 3.2 of [2], Aggarwal and Wee show that the matrix M discussed above is totally monotone and that by using the algorithm in [1] we can find the row minimas in $O(r+q)$ time. ∎

After finding the minimum entry in each row of M, say $m(i,j)$ is such an entry, we check whether $m(i,j)$ is less than the distance between u_i and its West-South nearest neighbour in U. If so, then (u_i, v_j) is a restricted pair for U and V.

5.2 Analysis

Let $T(n)$ denote the time to compute WS(S) for the n given points. Also, let $D(r,q)$ denote the time to compute all restricted pairs for U and V. It can be verified that $T(n) < 2T(n/2) + D(r,q)$, and $T(2) = c$. Now in computing the restricted pairs, we convert it to a totally monotone matrix. Here we note that the entire matrix need not be computed and stored in advance. The matrix elements are actually distances between two points and can be computed on the fly in $O(1)$ time each. Also computing the row minimas takes $O(r+q)$ time [1]. The operations in the end, of checking the WS nearest neighbour in U, do not take more than $O(r+q)$ time. Hence $D(r,q) = O(r+q) = O(n)$. Therefore, we have the following :

Theorem 5.6 *Given a set S of planar points, in the L_p metric, $1 < p < \infty$, ARNN(S) can be computed in $O(n \log n)$ time.*

Chapter 6

Conclusions

We propose algorithms to solve the SARNN problem optimally in $O(n \log n)$ time in the L_1 and the L_∞ metrics using the plane sweep technique. We show that with slight modifications, the above algorithms can be used to solve the ARNN problem in the L_1 and the L_∞ metrics also in $O(n \log n)$ time.

We also propose some modifications to Aggarwal and Wee's algorithm for the SARNN problem to optimally solve the ARNN problem for any L_p metric, $1 < p < \infty$, in $O(n \log n)$ time. The earlier algorithm of [6] takes $O(n \log^2 n)$ time.

All these algorithms are cost optimal, as it can be easily shown that $\Omega(n \log n)$ is the lower bound for these problems (which follows from the lower bound for closest pair/element distinctness [9]). However, it may be possible to further simplify them.

In addition, we discuss parallelization of the SARNN problem using the monotone matrix method. If an $O(\log n)$ time, optimal parallel algorithm is discovered for the monotone matrix problem, we show that this will also result in an optimal parallel algorithm for the SARNN problem which will take $O(\log^2 n)$ time and does $O(n \log n)$ work.

Discovering an optimal parallel algorithm directly for SARNN problem is open. Discovering an optimal parallel algorithm for monotone matrix problem is, in itself, an interesting problem.

Appendix A

Complete Algorithm for the SARNN problem in the L_∞ metric

Here, too we make the same assumptions as in Chapter 3. To find the SARNN graph of the ENWS region pair, in the L_∞ metric, we do the following :

First, we sort the points by their X-coordinates. The sweep line is vertical and proceeds from right to left. We assume that the sweep line intersects only 1 point at a time i.e. no two points have the same X-coordinate. At each event point, we find the set of points on the data structure which could be the SARNN for the event point. i.e. which lie in the EN region of the event point. This can be checked by the following 2 conditions :
1) $y_j > y_i$ and 2) $(y_i - x_i) > (y_j - x_j)$ where the event point is (x_i, y_i) and the point to be checked is (x_j, y_j).

Each such point is deleted from the structure after checking its L_∞ distance to the current event point. After all the points in the EN region of the event point have been processed, the point with the least L_∞ distance from the event point is set as its SARNN. Then, the event point is added to the data structure and the sweep line proceeds to the next event point. This is done until the sweep line goes past the last point. At this stage, the points still on the data structure are deleted as they don't have a neighbour in the WS direction. The correctness of this method follows from Lemma 3.2.

Appendix B

Complete Algorithm for the ARNN problem in the L_1 metric

Here, we assume that $k = 8$. Hence, the plane around each point is to be divided into eight regions. Now here, we don't pair regions like in the SARNN problem. Instead, the graph for each region is found separately and then merged later on to get the complete ARNN graph. To find the ARNN graph of the WS region, we do the following :

We first sort the points in the direction of the line with slope 1. i.e sort them by the sum $(y_i + x_i)$. Then we proceed from the point which has greatest value of $(y_i + x_i)$ towards the point which has lowest value of $(y_i + x_i)$. The sweep line has slope -1. We assume that the sweep line intersects only 1 point at a time. At each event point, we find the set of points on the data structure which could be the ARNN for the event point. i.e. which lie in the EN region of the event point. This can be checked by the following 2 conditions :
1) $y_j > y_i$ and 2) $(y_i - x_i) > (y_j - x_j)$ where the event point is (x_i, y_i) and the point to be checked is (x_j, y_j).

Each such point is deleted from the structure after setting its ARNN in the WS direction to be the current event point. After all the points in the EN region of the event point have been looked at and deleted, the event point is added to the data structure and the sweep line proceeds to the next event point. This is done until the sweep line goes past the last point. At this stage, the points still on the data structure don't have a neighbour in the WS direction and hence they can be deleted. The correctness of this method follows from Lemma 3.1.

We can find the ARNN graphs of the rest of the region by suitable rotation of the axes. These graphs are then merged to get the complete ARNN graph.

Appendix C

Complete Algorithm for the ARNN problem in the L_∞ metric

Here, too we make the same assumptions as in Appendix B. To find the ARNN graph of the WS region in the L_∞ metric, we do the following :

First, we sort the points by their X-coordinates. The sweep line is vertical and proceeds from right to left. We assume that the sweep line intersects only 1 point at a time i.e. no two points have the same X-coordinate. At each event point, we find the set of points on the data structure which could be the ARNN for the event point. i.e. which lie in the EN region of the event point. This can be checked by the following 2 conditions :
1) $y_j > y_i$ and 2) $(y_i - x_i) > (y_j - x_j)$ where the event point is (x_i, y_i) and the point to be checked is (x_j, y_j).

Each such point is deleted from the structure after setting its ARNN in the WS direction to be the current event point. After all the points in the EN region of the event point have been processed, the event point is added to the data structure and the sweep line proceeds to the next event point. This is done until the sweep line goes past the last point. At this stage, the points still on the data structure are deleted as they don't have a neighbour in the WS direction. The correctness of this method follows from Lemma 3.2.

Bibliography

[1] AGGARWAL, A., KLAWE, M. M., MORAN, S., SHOR, P., AND WILBER, R. Geometric applications of a matrix searching algorithm. In *Proceedings of the Symposium on Computational Geometry* (1986), pp. 285–292.

[2] AGGARWAL, A., AND WEE, Y. On the symmetric angle-restricted nearest neighbor problem. *IPL: Information Processing Letters 92* (2004), 121–126.

[3] ATALLAH, M., AND KOSARAJU, S. An efficient parallel algorithm for the row minima of a totally monotone matrix. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms* (1991), pp. 394–403.

[4] BRADFORD, P., FLEISCHER, R., AND SMID, M. More efficient parallel totally monotone matrix searching. *Journal of Algorithms 23* (1997), 386–400.

[5] CORMEN, T., STEIN, C., RIVEST, R., AND LEISERSON, C. *Introduction to Algorithms.* Prentice-Hall of India Pvt. Ltd., 2001.

[6] GUIBAS, L., AND STOLFI, J. On computing all north-east nearest neighbors in the l_1 metric. *Information Processing Letters 17*, 4 (1983), 219–223.

[7] JAJA, J. *Introduction to Parallel Algorithms.* Addison-Wesley, New York, 1992.

[8] L_p metrics. *http://msl.cs.uiuc.edu/planning/node177.html* .

[9] PREPARATA, F., AND SHAMOS, M. *Computational Geometry: An Introduction.* Springer-Verlag, 1985.

[10] Raman, R., and Vishkin, U. Optimal randomized parallel algorithms for computing the row maxima of a totally monotone matrix. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms* (1994), pp. 613–621.

[11] Wee, Y., Chaiken, S., and Willard, D. On the angle restricted nearest neighbor problem. *Information Processing Letters 34*, 2 (1990), 71–76.